Microsoft

# Kubernetes By Example

@rorypreddy

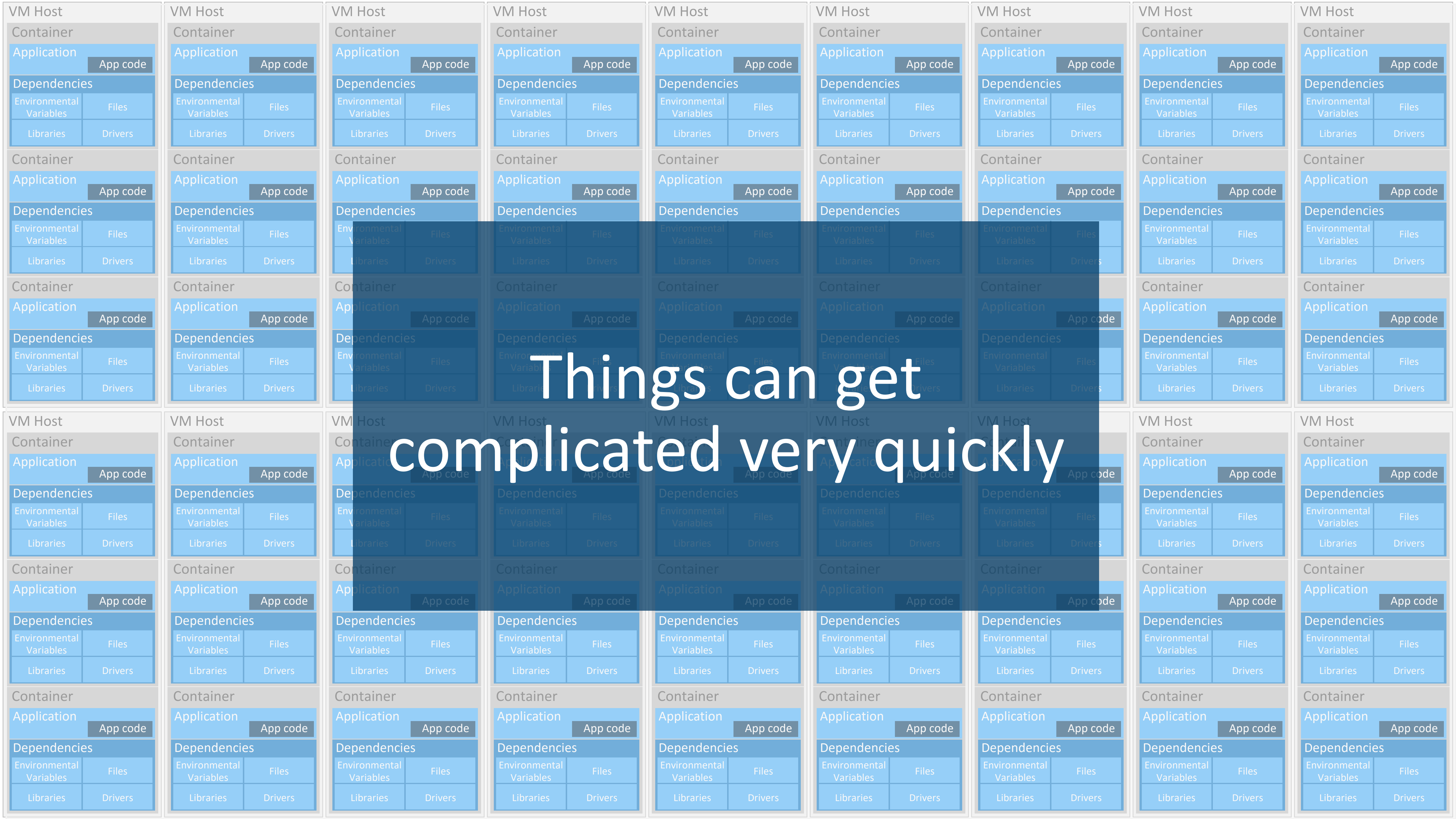
https://aka.ms/Kubernetes-java

# Agenda

- Why orchestration is needed?
- What is Kubernetes?
- The history and evolution of Kubernetes
- Kubernetes architecture and components
- Toolset
- Demo

Microsoft

# Container Challenges



# Things can get complicated very quickly

# Container Management at Scale

**Cluster Management:** deploy and manage cluster resources

**Scheduling:** where containers run

**Lifecycle and Health:** keep containers running despite failure

**Naming and Discovery:** where are my containers

**Load Balancing:** evenly distribute traffic

At the end of the day we need something to help us with all the orchestration..
An orchestrator!

**Scaling:** make se
containe
in numb

secure Docker
container images

and workflow

happening in
containers and
cluster

lata
for containers

# Available Orchestrators

- Docker Swarm
- Apache Mesos
- Nomad (from HashiCorp)
- Rancher
- Service Fabric
- …
- Kubernetes

MESOS

HashiCorp
Nomad

RANCHER

Microsoft Azure
Service Fabric

Microsoft

# Kubernetes is Born

# What is Kubernetes (k8s)?

- **Kubernetes** is "an open-source software for automating deployment, scaling, and management of containerized applications“.
- **Kubernetes**, in Greek κυβερνήτης, means the Helmsman, or pilot of the ship.
- Keeping with the maritime theme of **Docker** containers, **Kubernetes** is the pilot of a ship of containers.

# What is Kubernetes (k8s)?

**History**

- Originally designed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF).
- Google still actively involved
- Kubernetes v1.0 was released on July, 2015 by Joe Beda, Brendan Burns and Craig McLuckie
- Most discussed repo in GitHub last year.
- Over 1,700 authors and releases every three month

- To learn more about the ideas behind Kubernetes: read the Large-scale cluster management at Google with Borg paper

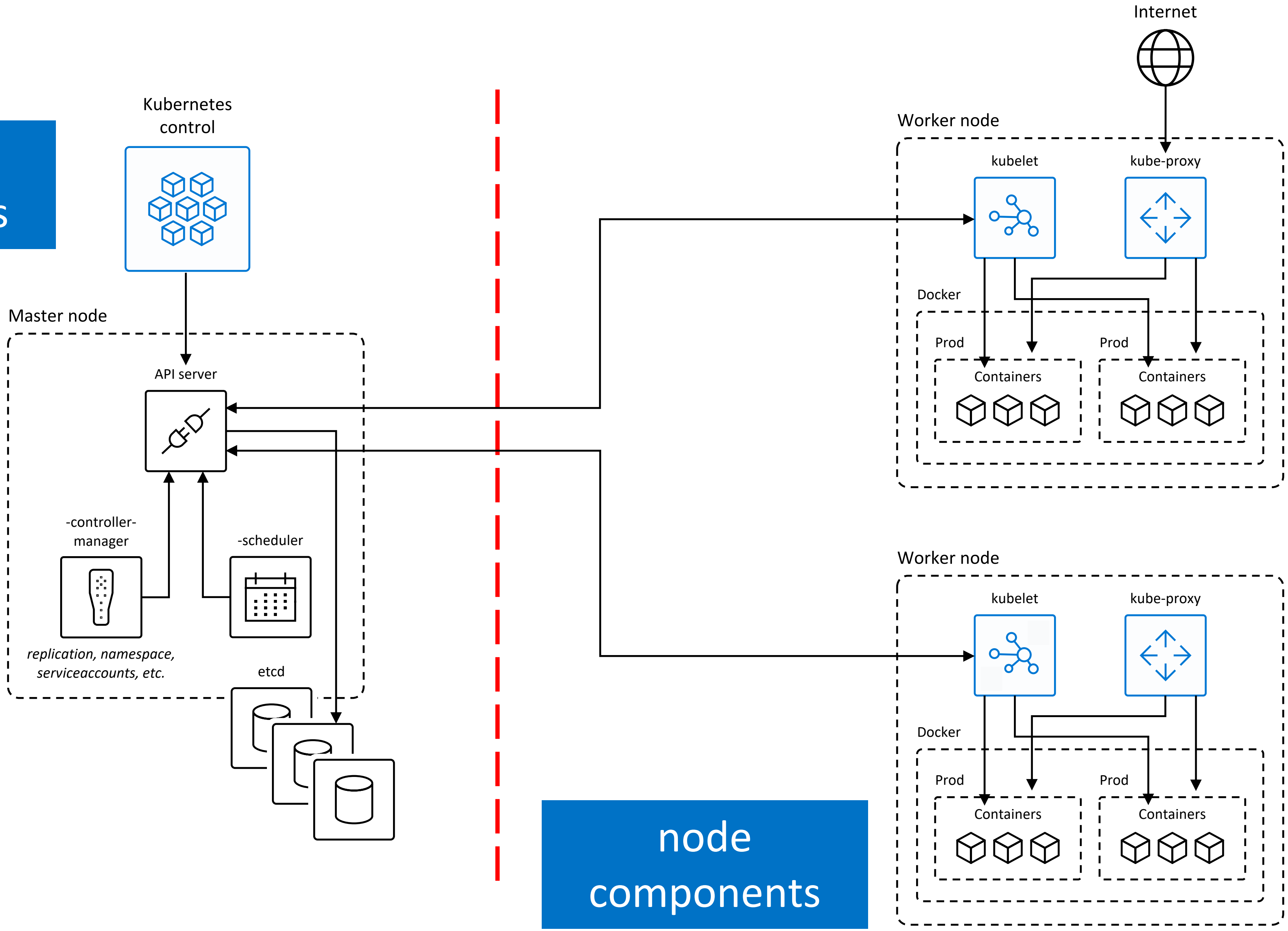
# Kubernetes Features

- Self-Healing
- Horizontal Scaling
- Automated rollouts and rollbacks
- Service Discovery and Load Balancing
- Automatic bin packing
- Storage orchestration
- Secret and configuration management

Microsoft

# Kubernetes Building Blocks

## master components

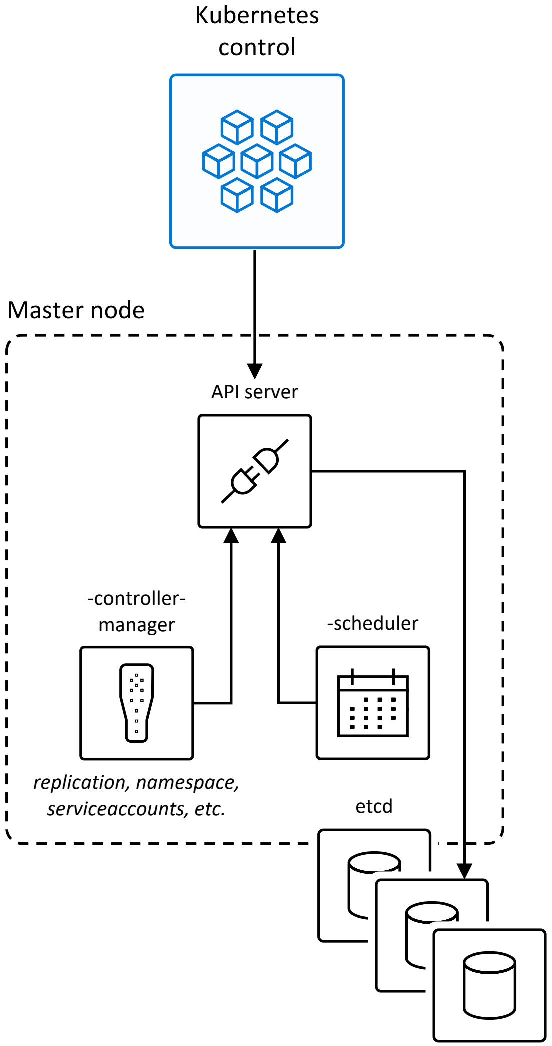


## node components

# kubectl

master components

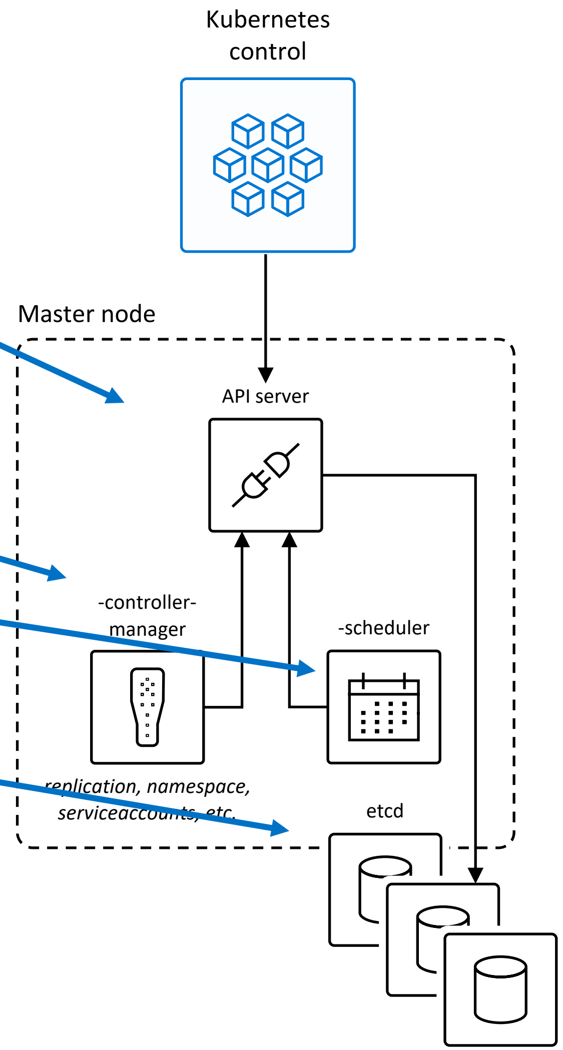
- CLI to run commands against a Kubernetes cluster
  - Swiss Army Knife: run deployments, exec into containers, view logs, etc.
  - Pronounced "koob sea tee el" or "koob cuddle"
  - Available for Windows and Linux – of course available in Azure Cloud Shell



master components

# Master Components

- **Kube-api-server**
  - Front-end control plane. Exposes API
- **controller-manager**
  - Runs controllers, e.g. replication controller, node controller
- **scheduler**
  - assigns pods to nodes
- **etcd**
  - Highly available, distributed Cluster database.
- **add-ons**
  - DNS, Heapster (enables monitoring and performance analysis), Dashboard, Logging

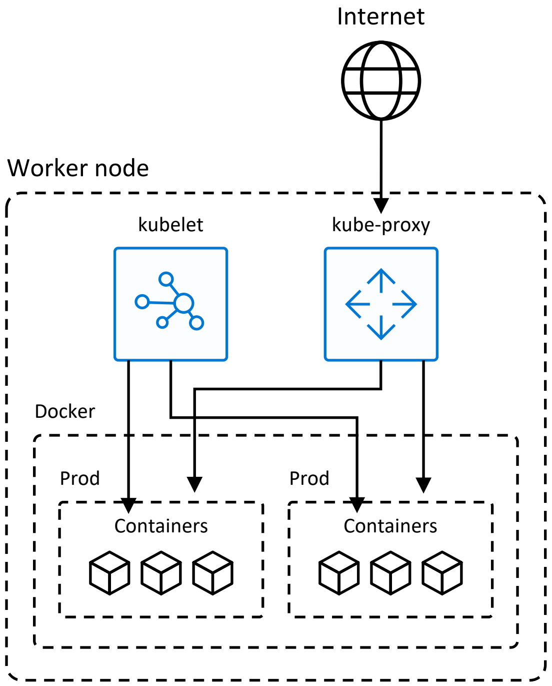


# Worker Node Compos

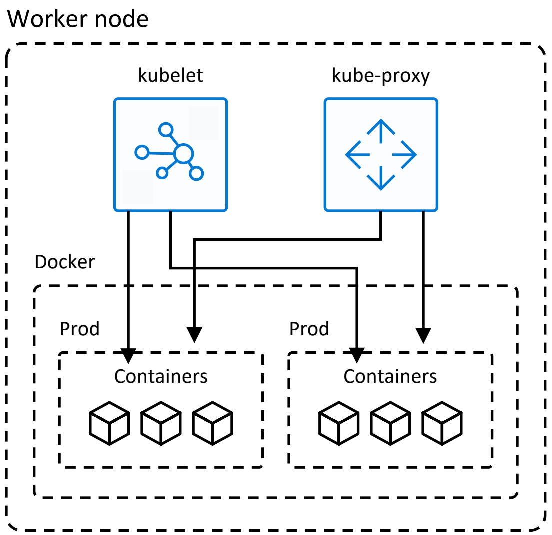
- **Kubelet**
  - Primary node agent
  - Watches and runs assigned pods
  - Executes health probes and reports status

- **Kube-proxy**
  - Enables network services

- **Container Runtime**
  - Docker, rkt, runc and other OCI implementations..



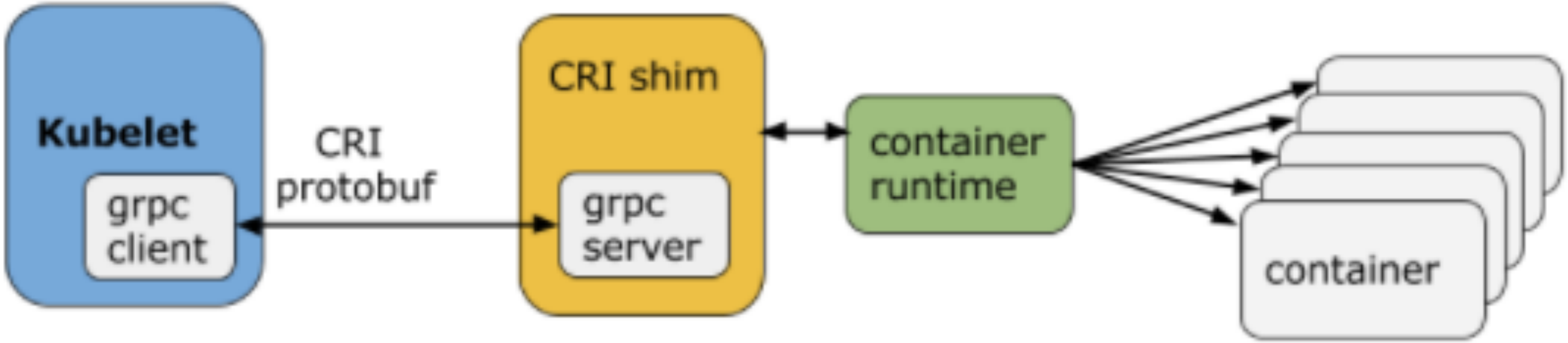
node components



# Worker Node Components

- **Container Runtime**
  - Kubernetes has **no code** to execute or run containers on Linux or Windows
  - Initially the Kubernetes pod manager (called "kubelet") had direct linkage to the **Docker** engine

  - Container Runtime Interface (CRI) was introduced with Kubernetes 1.5



# Worker Node Components

- Container Runtime Interface (CRI)
  - provides a clearly-defined abstraction layer
  - eliminates barriers building own container runtimes
  - enabling pluggable container runtimes

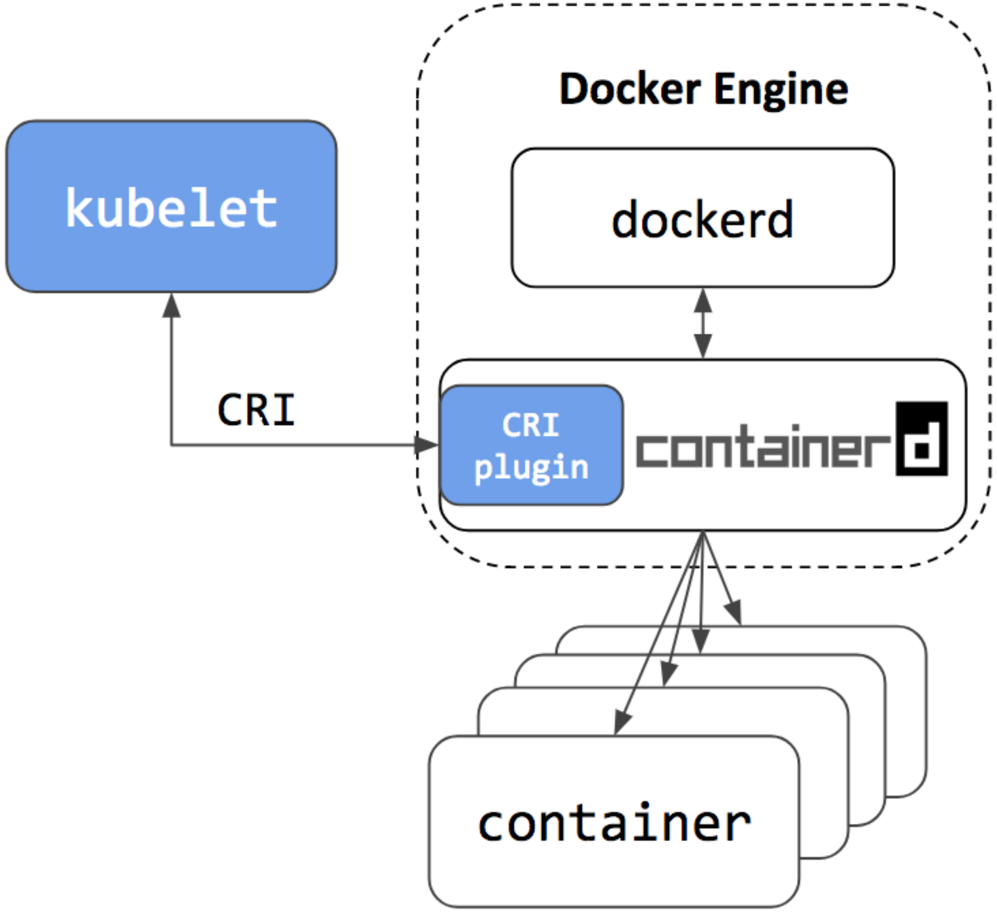
More information about CRI can be found at -

https://kubernetes.io/blog/2016/12/container-runtime-int

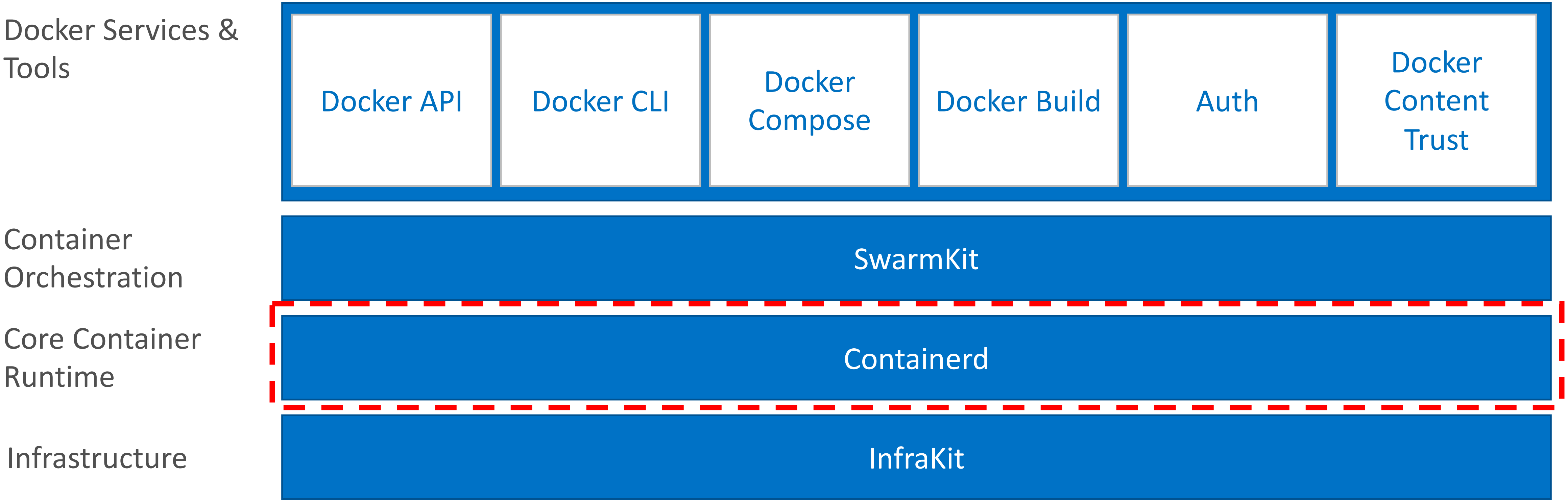
in-kubernetes/

Runtimes supported today, either upstream or by forks, inc

docker (for Linux and Windows), rkt, cri-o, and frakti.



# Docker Engine



Docker Services & Tools

Docker API

Docker CLI

Docker Compose

Docker Build

Auth

Docker Content Trust

Container Orchestration

SwarmKit

Core Container Runtime

Containerd

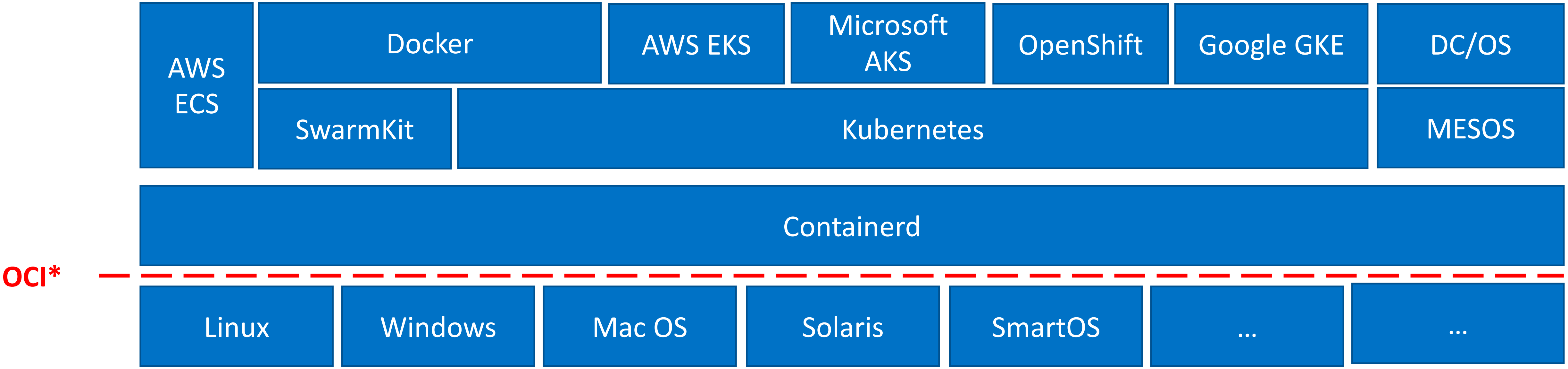
Infrastructure

InfraKit

DOCKER TO DONATE CONTAINERD TO THE CLOUD NATIVE COMPUTING FOUNDATION (MARCH 2017)

https://blog.docker.com/2017/03/docker-donates-containerd-to-cncf/

# Containerd’s role in Container Ecosystem



AWS ECS

Docker

AWS EKS

Microsoft AKS

OpenShift

Google GKE

DC/OS

SwarmKit

Kubernetes

MESOS

Containerd

OCI*

Linux

Windows

Mac OS

Solaris

SmartOS

…

…

**containerd deep dive @ containerd summit for more**
https://www.youtube.com/watch?v=UUDDCetB7_A

*OCI = Open Container Interface

# What is Moby?

- **Containerd** – core container runtime
- **Linuxkit** – tool to build secure, portable and lean os for containers.
- **Infrakit** – creating and managing self healing infrastructures.
- **Libnetwork** – native Go implementation to connect containers.

**About Moby:**
“An open framework to assemble specialized container systems without reinventing the wheel.”

mobyproject.org

**Why do we care?**
“To deliver more frequent upstream patches and improvements to the container runtime, AKS has adopted Moby, the open-source project that Docker is based on. “

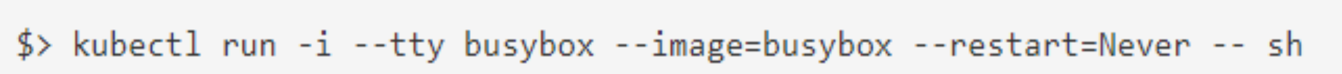
https://azure.microsoft.com/en-us/updates/azure-kubernetes-service-december-update/

Microsoft

# Kubernetes Resources

# Declarative vs. Imperative

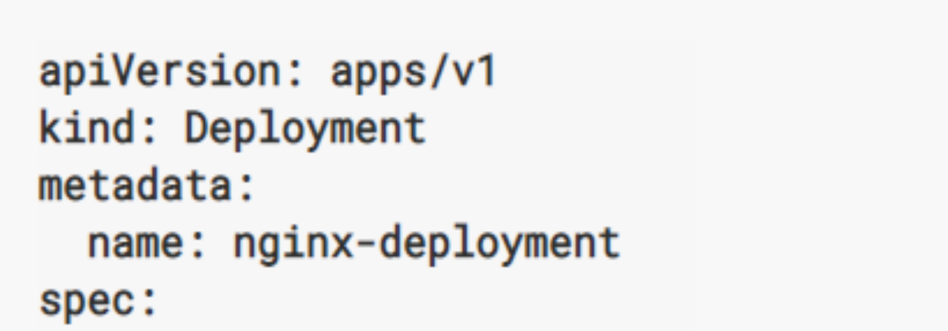
- Commands like kubectl run and kubectl expose are **imperative** commands (do this thing now)



```
$> kubectl run -i --tty busybox --image=busybox --restart=Never -- sh
```

- **Declarative** way – Describe the state of resources in a file (JSON or YAML).

  kubectl apply –f webresource.yaml



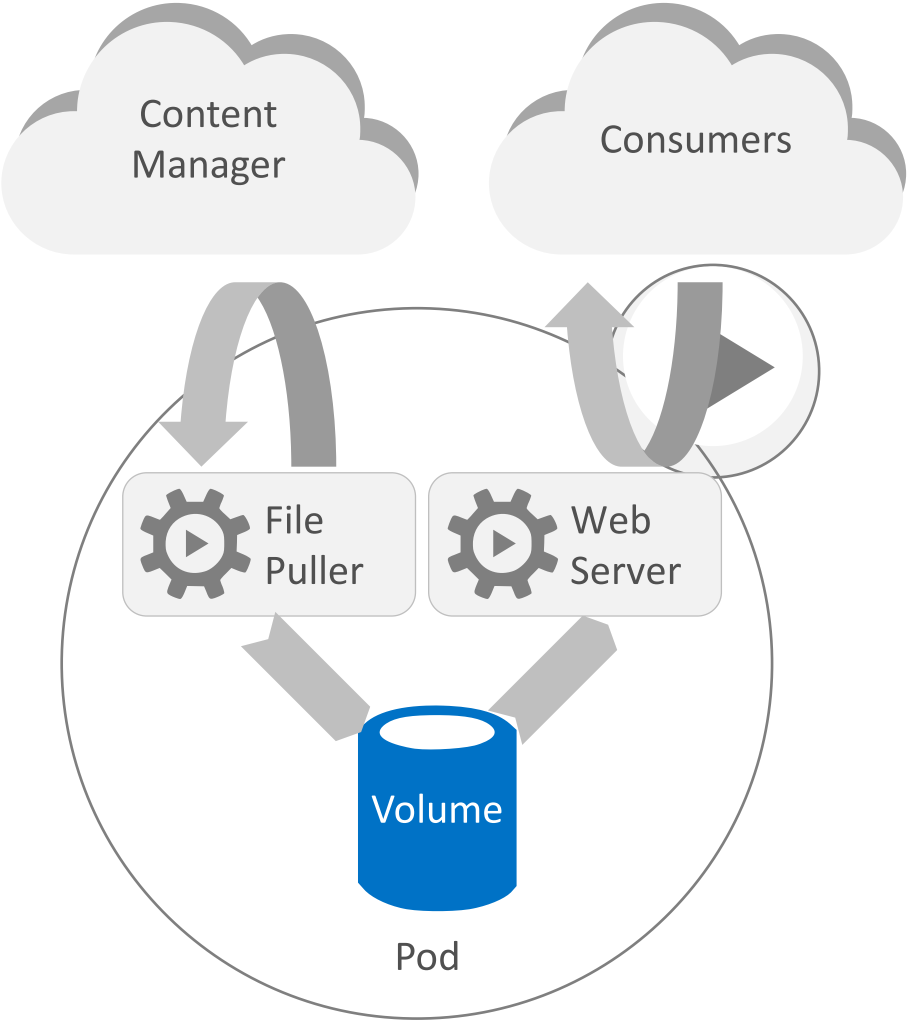
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
```

# Kubernetes Resources

| | | |
|---|---|---|
| pod | ReplicaSet | deployment |
| service | namespace | volumes |
| config-map | secret | ingress |
| StatefulSet | DaemonSet | jobs |

# What is a pod?

- Pod is the basic building block in Kubernetes
- Pods are how containers are delivered
- Can be multiple containers (e.g. side car)
- Encapsulates container(s), storage, network IP, and options on how to run



# What is a pod? Considerations..

- Pods do not, by themselves, self-heal!
  - If a Pod is scheduled to a Node that fails, or if the scheduling operation itself fails, the Pod is deleted; likewise, a Pod won’t survive an eviction due to a lack of resources or Node maintenance.
- Thus, while it is possible to use Pod directly, it’s far more common in Kubernetes to manage your pods using a Controller.
- Controllers can create and manage multiple Pods, handling replication and rollout and providing self-healing capabilities at cluster scope.
  - For example, if a Node fails, the Controller might automatically replace the Pod by scheduling an identical replacement on a different Node.

# What is a pod?

- Examples of Controllers that contain one or more pods:
  - StatefulSet
  - DaemonSet
  - ReplicaSet (+ Deployments)
  - Jobs

deployment

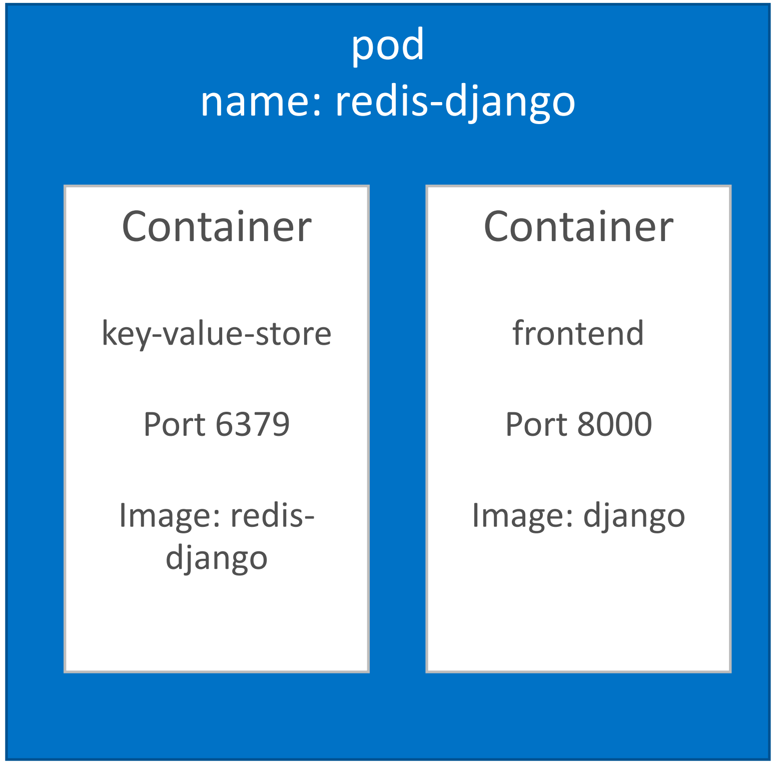
StatefulSet

DaemonSet

ReplicaSet

# Kubernetes manifest: Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: redis-django
  labels:
    app: web
spec:
  containers:
    - name: key-value-store
      image: redis
      ports:
        - containerPort: 6379
    - name: frontend
      image: django
      ports:
        - containerPort: 8000
```



pod
name: redis-django

Container

key-value-store

Port 6379

Image: redis-django

Container

frontend

Port 8000

Image: django

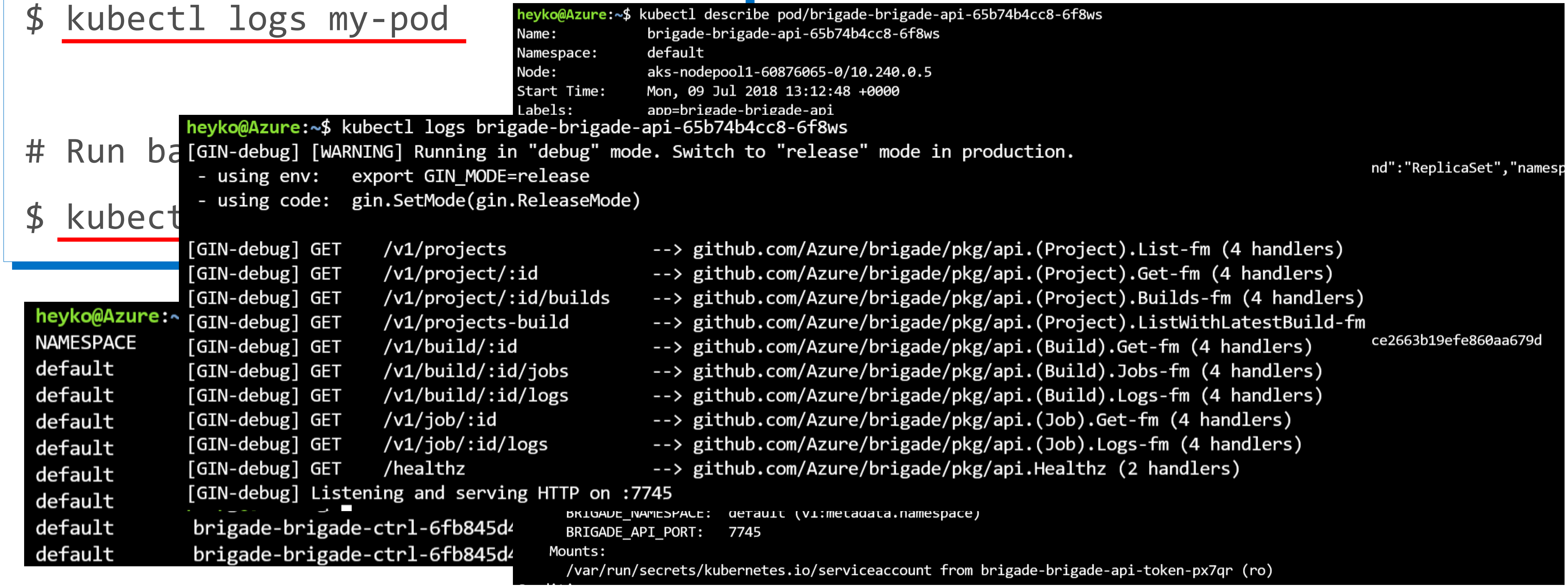
# Kubernetes manifest: Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: redis-django
  labels:
    app: web
spec:
  containers:
  - name: key-value-store
    image: redis
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- **Requests** specify how much resources a container needs.
- **Limits** define how much resources a container can consume.

# Interact with pods

```
$ kubectl get pod --all-namespaces

$ kubectl describe pod/my-pod

$ kubectl logs my-pod


# Run ba

$ kubect
```



```
heyko@Azure:~$ kubectl describe pod/brigade-brigade-api-65b74b4cc8-6f8ws
Name:           brigade-brigade-api-65b74b4cc8-6f8ws
Namespace:      default
Node:           aks-nodepool1-60876065-0/10.240.0.5
Start Time:     Mon, 09 Jul 2018 13:12:48 +0000
Labels:         app=brigade-brigade-api
nd":"ReplicaSet","namesp
ce2663b19efe860aa679d
  BRIGADE_NAMESPACE:  default (v1:metadata.namespace)
  BRIGADE_API_PORT:   7745
 Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from brigade-brigade-api-token-px7qr (ro)
```

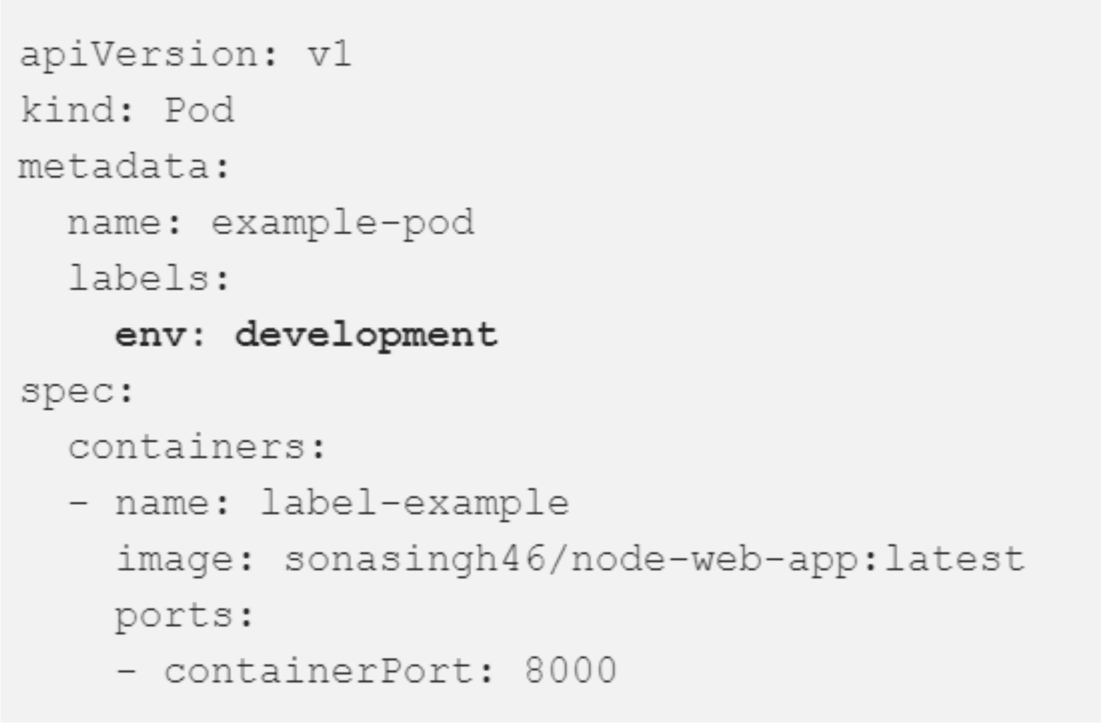
```
heyko@Azure:~$ kubectl logs brigade-brigade-api-65b74b4cc8-6f8ws
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /v1/projects              --> github.com/Azure/brigade/pkg/api.(Project).List-fm (4 handlers)
[GIN-debug] GET    /v1/project/:id           --> github.com/Azure/brigade/pkg/api.(Project).Get-fm (4 handlers)
[GIN-debug] GET    /v1/project/:id/builds    --> github.com/Azure/brigade/pkg/api.(Project).Builds-fm (4 handlers)
[GIN-debug] GET    /v1/projects-build        --> github.com/Azure/brigade/pkg/api.(Project).ListWithLatestBuild-fm
[GIN-debug] GET    /v1/build/:id             --> github.com/Azure/brigade/pkg/api.(Build).Get-fm (4 handlers)
[GIN-debug] GET    /v1/build/:id/jobs        --> github.com/Azure/brigade/pkg/api.(Build).Jobs-fm (4 handlers)
[GIN-debug] GET    /v1/build/:id/logs        --> github.com/Azure/brigade/pkg/api.(Build).Logs-fm (4 handlers)
[GIN-debug] GET    /v1/job/:id               --> github.com/Azure/brigade/pkg/api.(Job).Get-fm (4 handlers)
[GIN-debug] GET    /v1/job/:id/logs          --> github.com/Azure/brigade/pkg/api.(Job).Logs-fm (4 handlers)
[GIN-debug] GET    /healthz                  --> github.com/Azure/brigade/pkg/api.Healthz (2 handlers)
[GIN-debug] Listening and serving HTTP on :7745
```

```
heyko@Azure:~
NAMESPACE
default
default
default
default
default
default
default       brigade-brigade-ctrl-6fb845d4
default       brigade-brigade-ctrl-6fb845d4
```

# Labels and Selectors

- Labels are key/value pairs for any API object in Kubernetes
- ”Label selectors” == queries against labels to match objects

- Use cases:
  - Associating pods to a service
  - Pinning workloads to specific nodes
  - Selecting a subset of resources



```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    env: development
spec:
  containers:
  - name: label-example
    image: sonasingh46/node-web-app:latest
    ports:
    - containerPort: 8000
```

# Controllers - Deployment

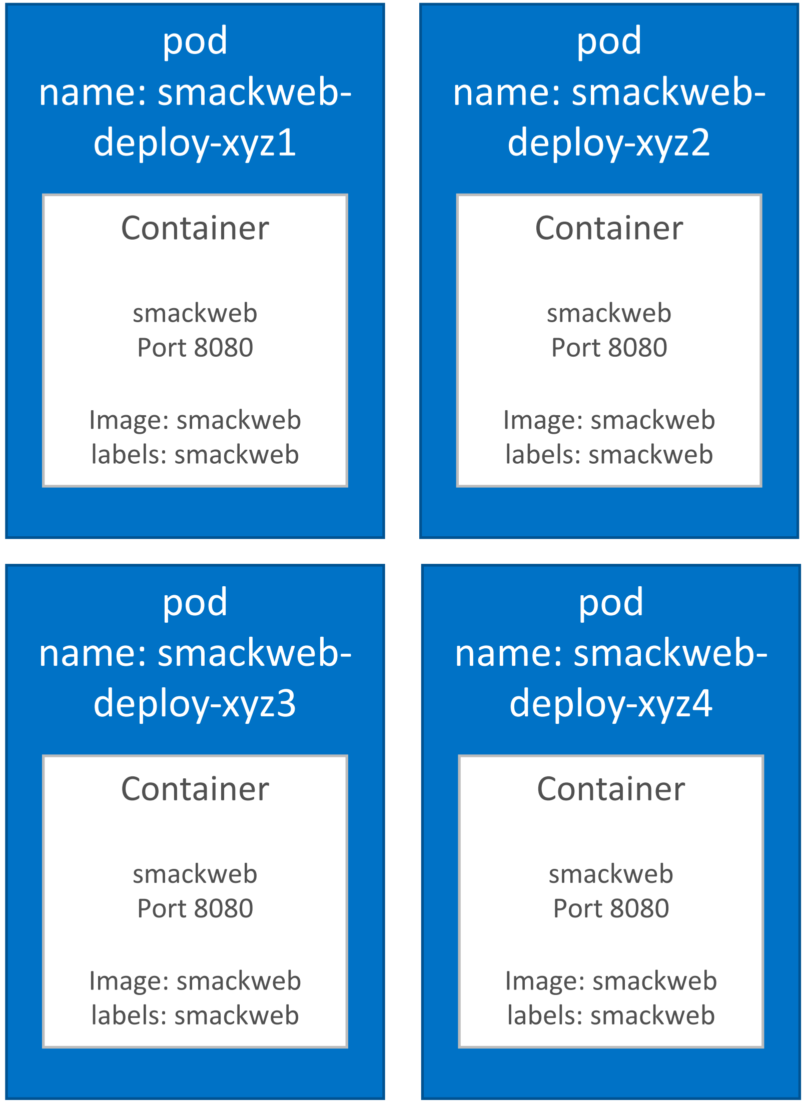
Deployment provides declarative updates for Pods and ReplicaSets.

Use Cases:

- Create deployment to rollout **ReplicaSet**
- Declare new state for pods (eg – new imageTag)
- Rollback to earlier revision
- Scale up or down
- Check rollout history
- Clean-up older ReplicaSets

# Kubernetes manifest: Deployment

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: smackweb-deploy
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: smackweb
    spec:
      containers:
      - name: smackweb
        image: chzbrgr71/smackweb
        ports:
        - containerPort: 8080
```



pod
name: smackweb-deploy-xyz1

Container

smackweb
Port 8080

Image: smackweb
labels: smackweb

pod
name: smackweb-deploy-xyz2

Container

smackweb
Port 8080

Image: smackweb
labels: smackweb

pod
name: smackweb-deploy-xyz3

Container

smackweb
Port 8080

Image: smackweb
labels: smackweb

pod
name: smackweb-deploy-xyz4

Container

smackweb
Port 8080

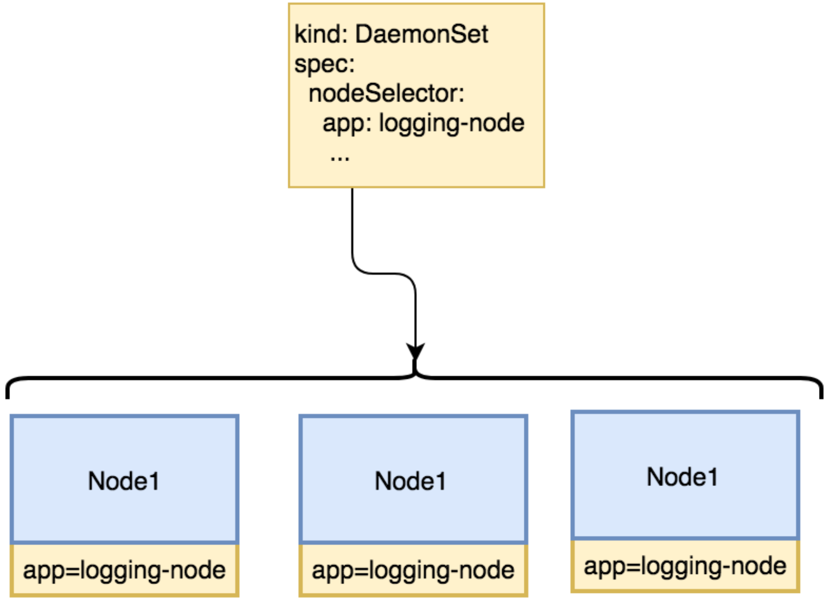
Image: smackweb
labels: smackweb

# Controllers - DaemonSets

DaemonSets ensure that all (or some) Nodes run a copy of a Pod.

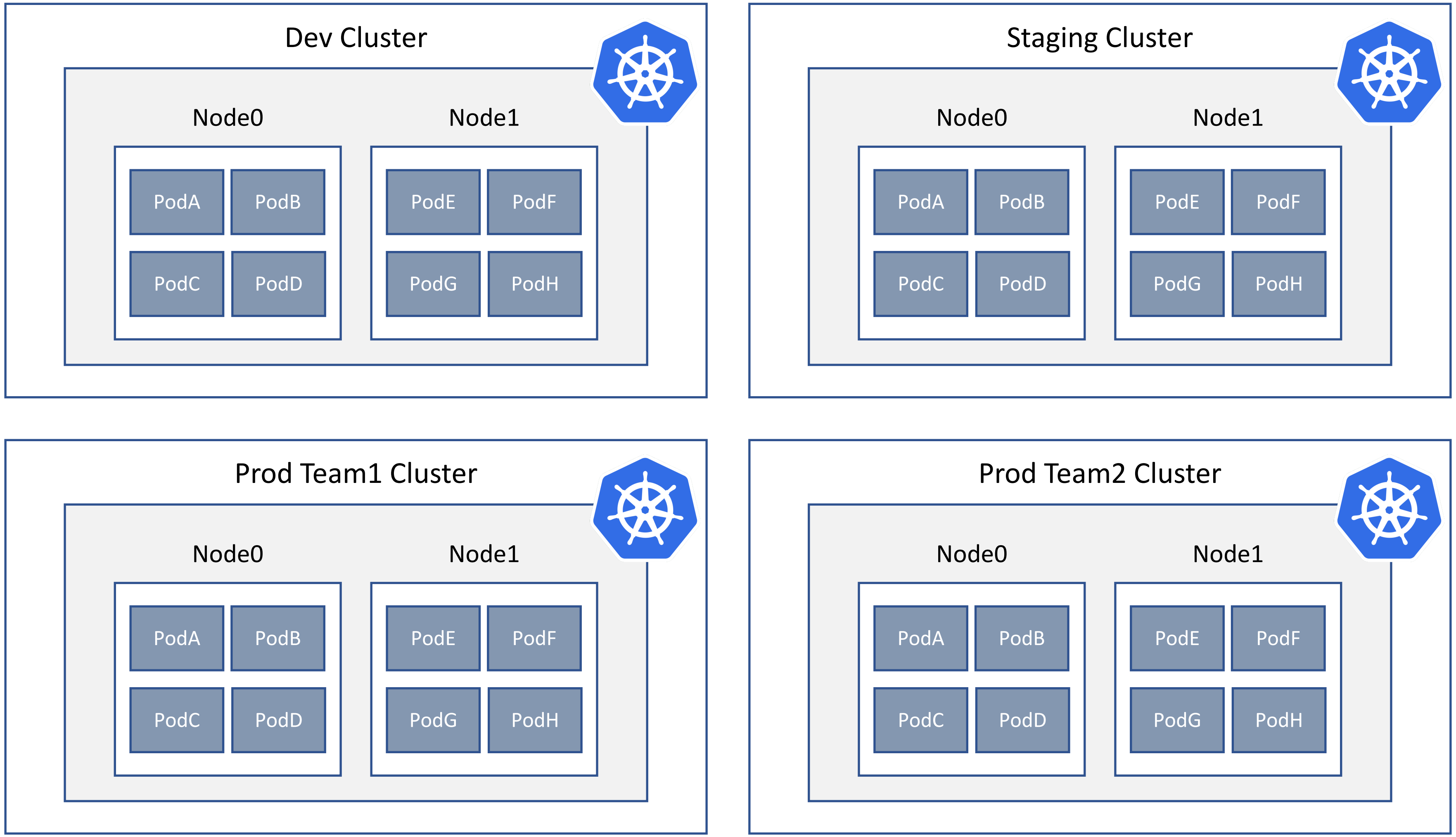
- As worker nodes are
  - added to the cluster, Pods are added to them.
  - removed from the cluster, those Pods are garbage collected.

- Some typical uses of a DaemonSet are:
  - logs collection daemon (i.e. fluentd, logstash)
  - Malware scan (install AV)
  - node monitoring daemon (i.e. Prometheus, collectd, Datadog, New Relic)

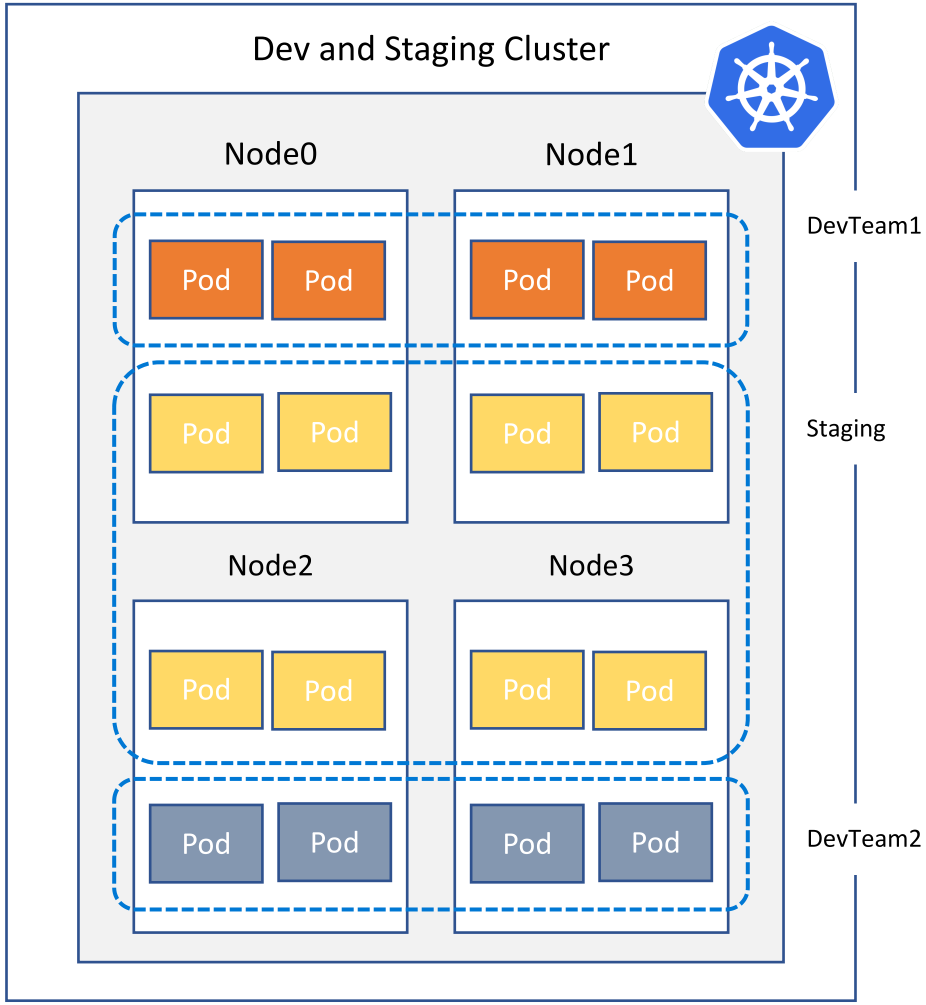


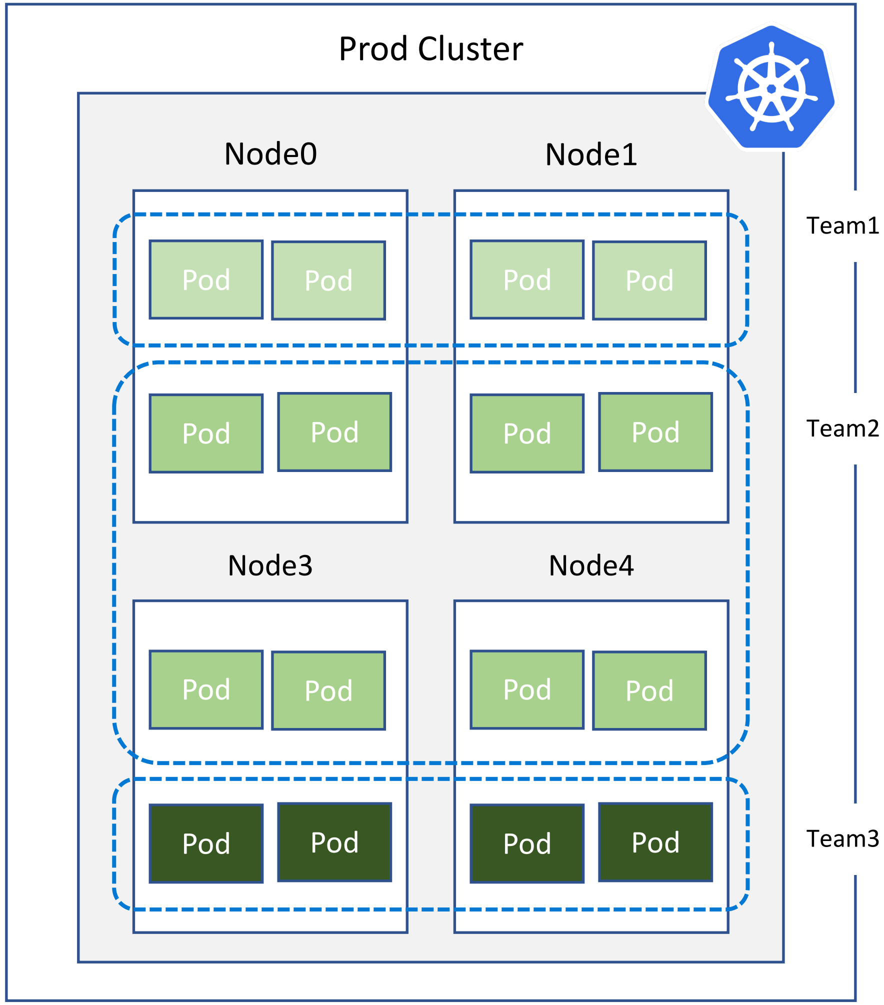
```
kind: DaemonSet
spec:
  nodeSelector:
    app: logging-node
   ...
```

# Cluster Isolation Patterns: Physical Isolation



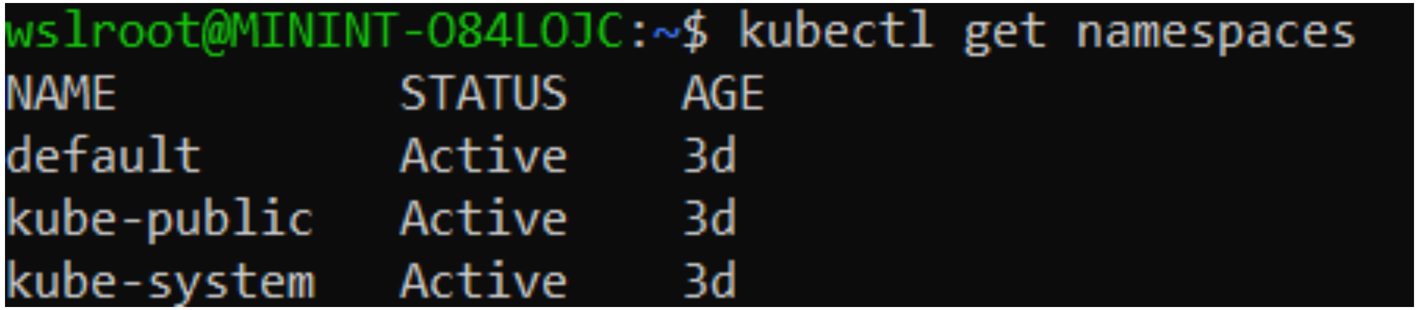
# Cluster Isolation Patterns: Logical Isolation





# Namespaces

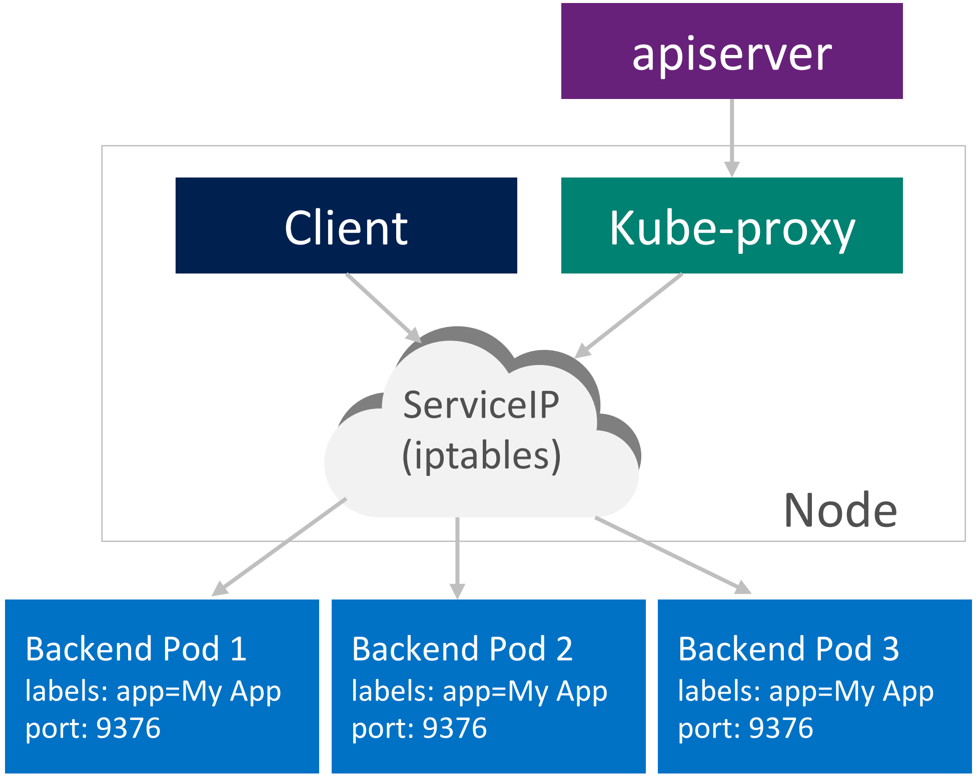
- multiple virtual clusters backed by the same physical cluster
- logical separation/isolation
- Every resource type is scoped to a namespace
  (except for nodes, persistentVolumes, etc.)
- Intended for environments with many users, teams, projects
- Kube-system namespace for dashboard etc.



```
wslroot@MININT-O84LOJC:~$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    3d
kube-public   Active    3d
kube-system   Active    3d
```

# Kubernetes Services

- Defines a logical set of pods
- Identified/selected using Labels

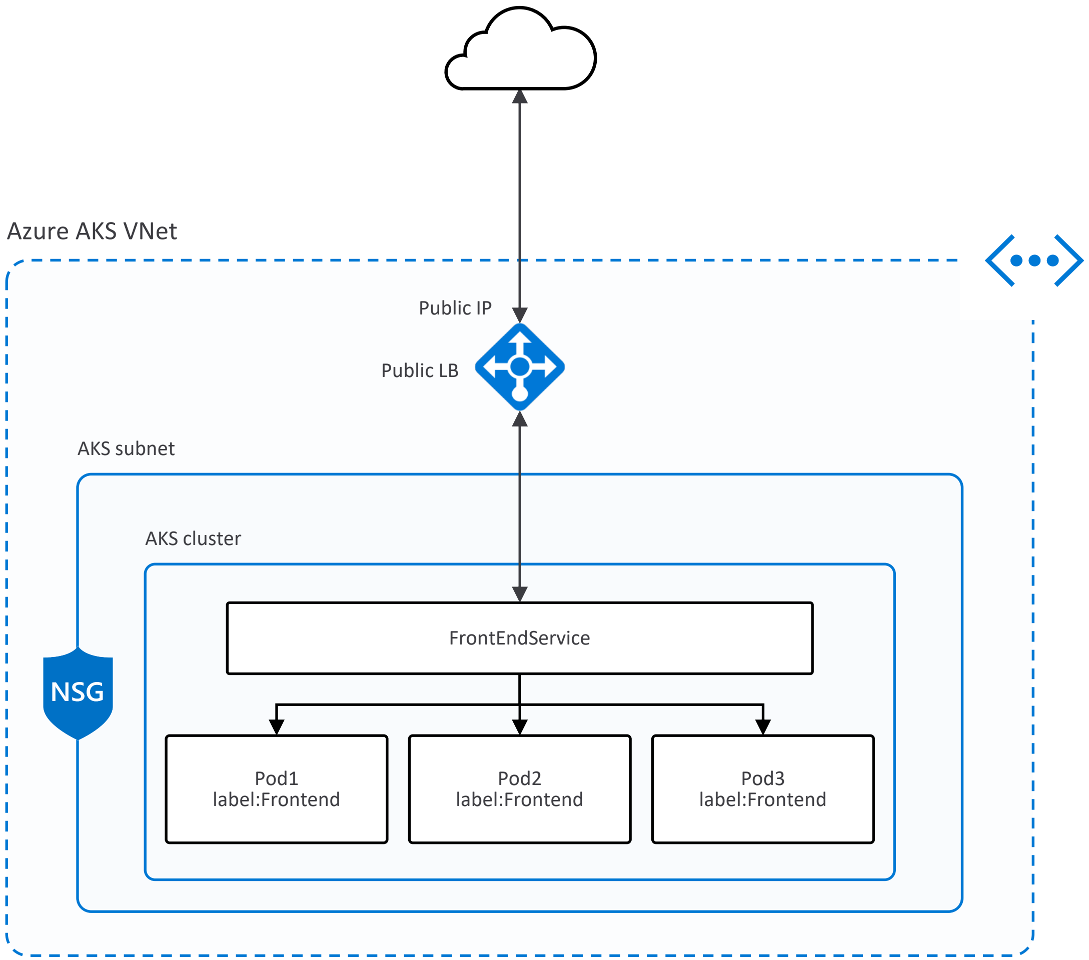


- Essentially a virtual load balancer in front of pods

# Public LoadBalancer Service

- Service Type LoadBalancer
- Basic Layer4 Load Balancing (TCP/UDP)
- Each service has assigned an IP on the ALB

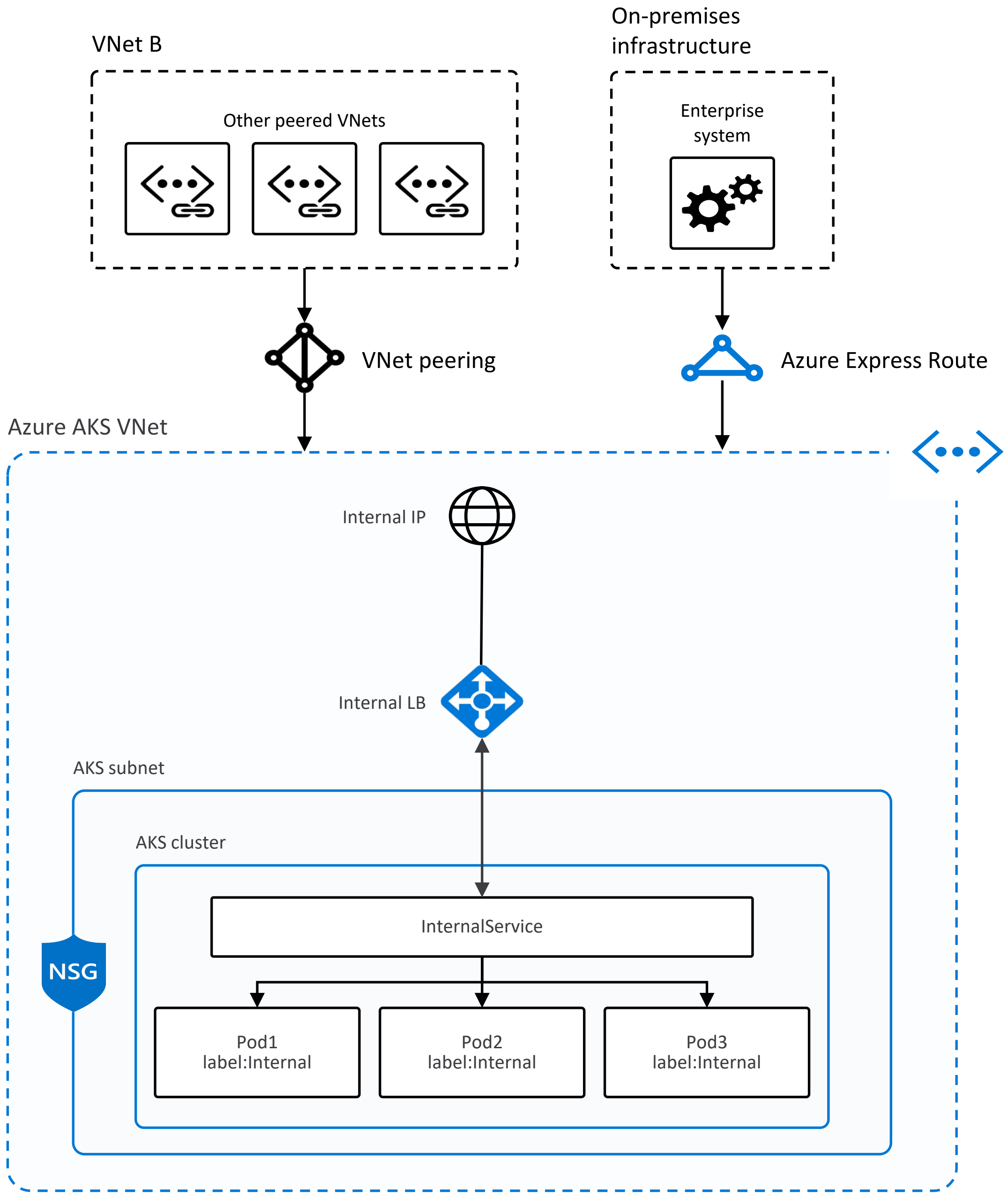
```yaml
apiVersion: v1
kind: Service
metadata:
   name: frontendservice
spec:
   loadBalancerIP: X.X.X.X
   type: LoadBalancer
   ports:
   - port: 80
   selector:
      app: frontend
```



# Internal LoadBalancer Service

- Used for internal services that should be accessed by other VNETs or On-Premise only

```yaml
apiVersion: v1
kind: Service
metadata:
  name: internalservice
  annotations:
    service.beta.kubernetes.io/azure–load–
balancer–internal: "true"
spec:
   type: LoadBalancer
   loadBalancerIP: 10.240.0.25
   ports:
   – port: 80
   selector:
    app: internal
```



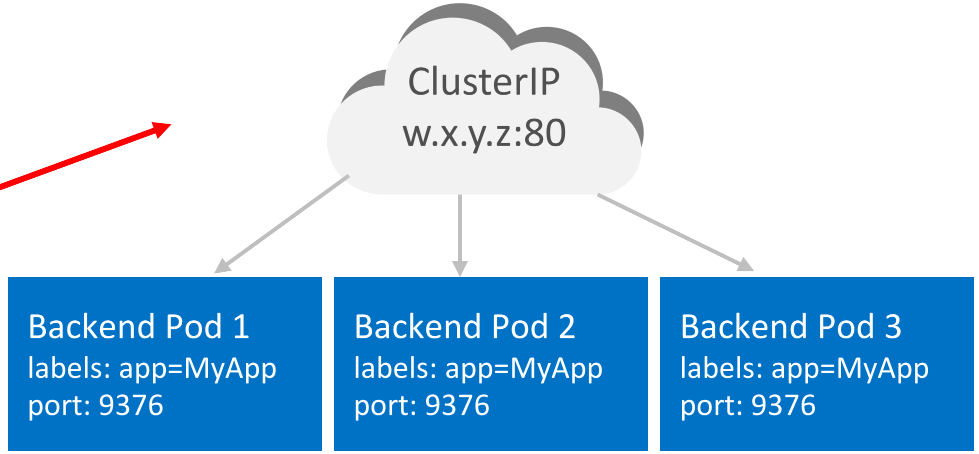
# Other Service Types

- **ClusterIP**
  - Exposes the service on a cluster-internal IP. Choosing this value makes the service **only reachable from within the cluster**

- **NodePort**
  - Exposes the service on each Node’s IP at a static port (the NodePort)
  - Connect from outside the cluster by requesting <NodeIP>:<NodePort>

# Kubernetes manifest: Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

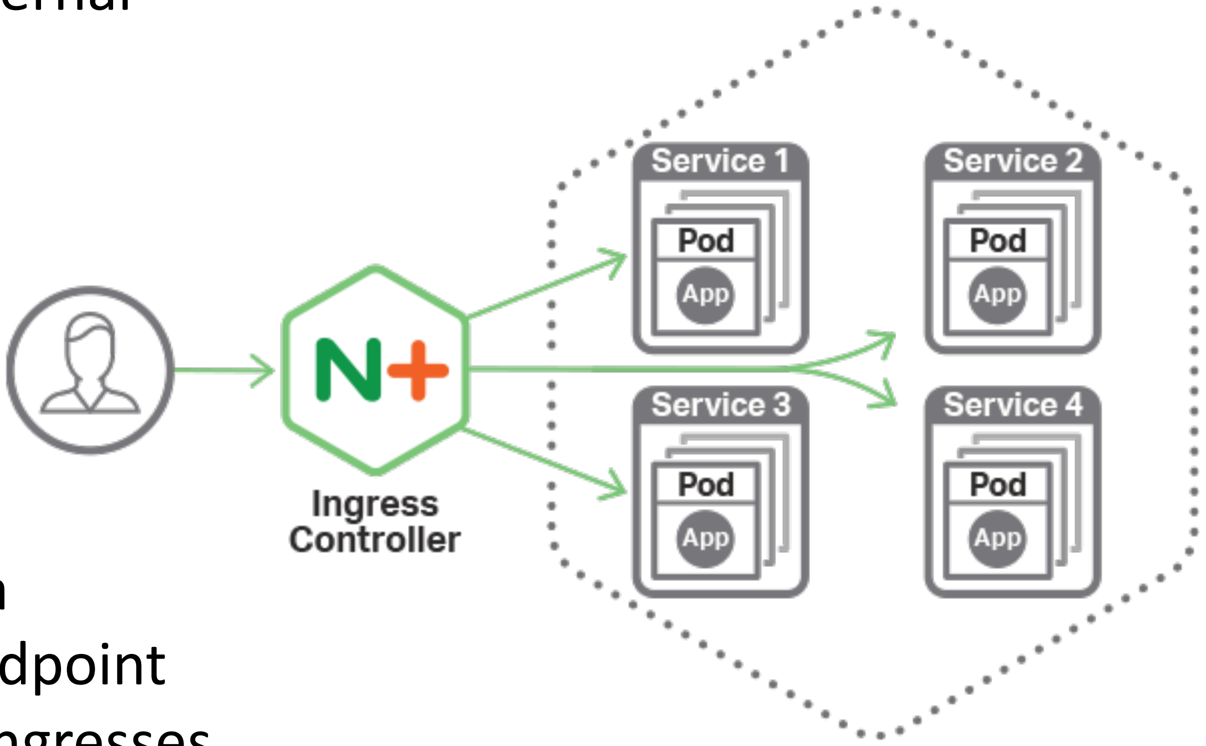


**Note!** Services using **ClusterIP** are only reachable from within the cluster.

# Ingress and Ingress Controllers

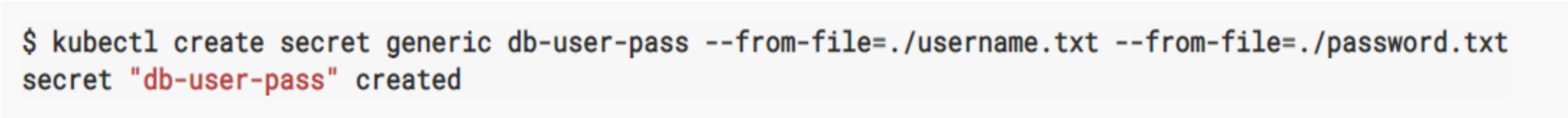
- Ingress is a Kubernetes API that manages external access to the services in the cluster
  - Supports HTTP and HTTPs
  - Path and Subdomain based routing
  - SSL Termination
  - Save on public Ips

- Ingress controller is a daemon, deployed as a Kubernetes Pod, that watches the Ingress Endpoint for updates. Its job is to satisfy requests for ingresses. Most popular one being Nginx.



# Secrets, Config Maps

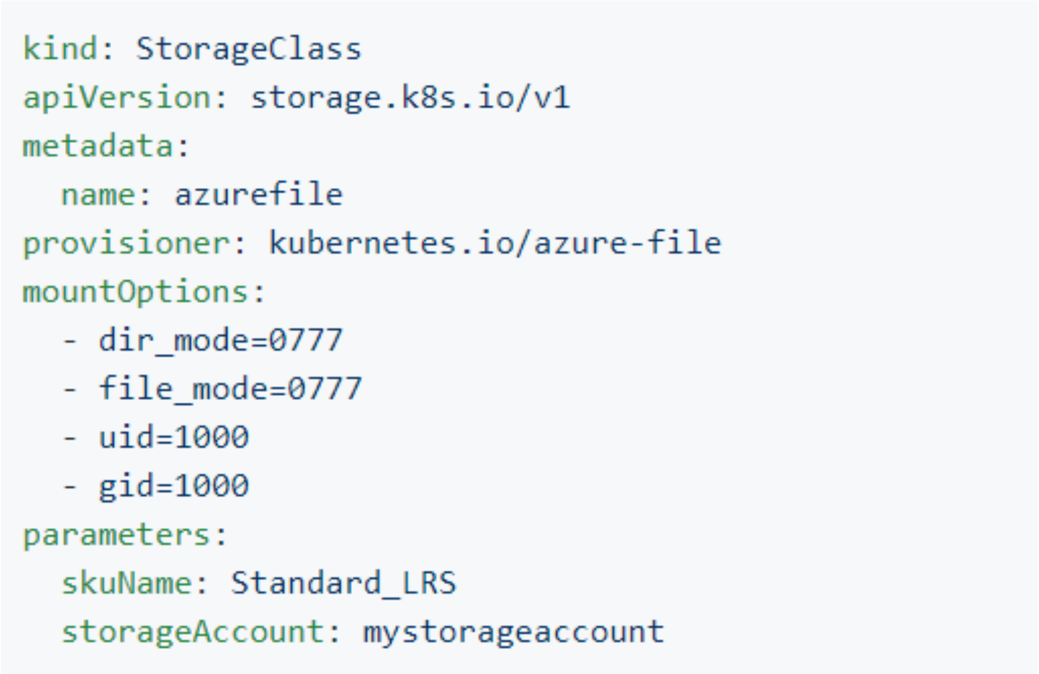
- **Secrets** are intended to hold sensitive information such as passwords, tokens. Secrets are for Confidential data. Secrets are encoded with Base64 encoding
- **ConfigMaps** help you to store non-confidential application configuration data. This helps to decouple configuration artifacts from image content



```
$ kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
secret "db-user-pass" created
```

# Kubernetes Volumes

- On disk files in a Container are ephermeral
- Files will be lost if Container crashes and then restarts
- Volumes outlive containers. Lifetime is same as that of a pod. Data is preserved across container restarts
- Persistent Volumes have lifetime independent of the Pod lifetime
- Types of volumes – emptydir, azureDisk, azureFile etc.

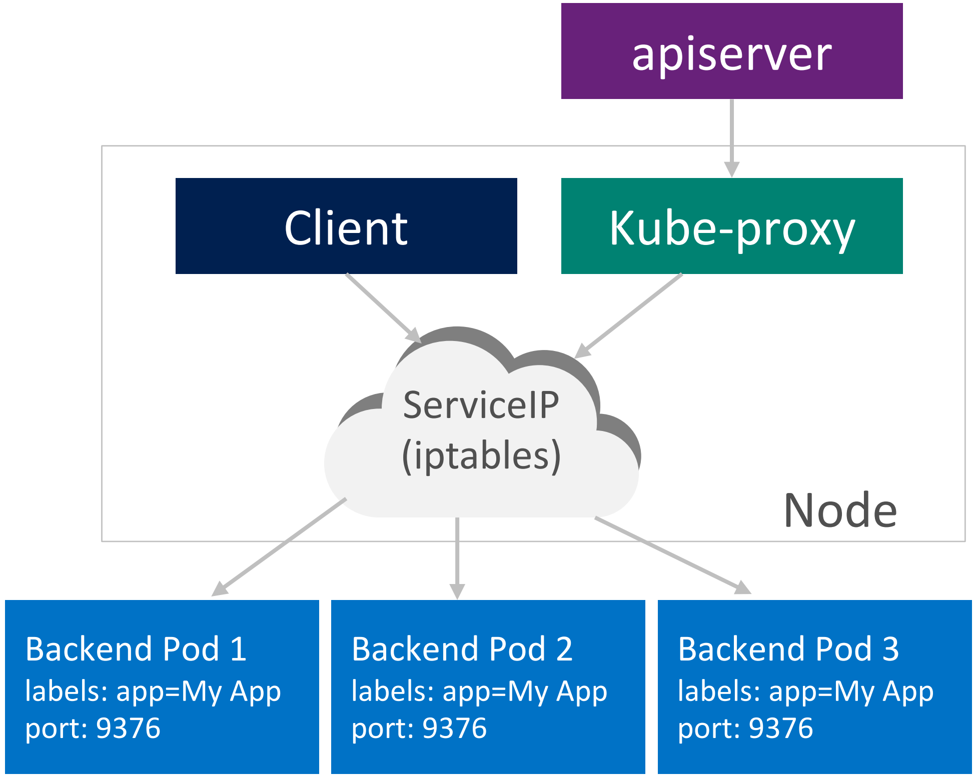


```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
  - uid=1000
  - gid=1000
parameters:
  skuName: Standard_LRS
  storageAccount: mystorageaccount
```

# Networking in Kubernetes

## Kubernetes knows 3 methods of communications:

- **Pod-to-Pod** communication directly by IP address. Kubernetes has a Pod-IP wide metric simplifying communication.
- **Pod-to-Service** Communication – Client traffic is directed to service virtual IP by kube-proxy process (running on all hosts) and directed to the correct Pod.
- **External-to-Internal** Communication – external access is captured by an external load balancer which targets nodes in a cluster. The Pod-to-Service flow stays the same.



# Demos, Labs and more

https://aka.ms/Kubernetes-java